An Adaptive Strategy for Scheduling Data-Intensive Applications in Grid Environments

Wantao Liu[1], Rajkumar Kettimuthu[3,4], Bo Li[1], Ian Foster[2,3,4]
[1] *School of Computer Science and Engineering, Beihang University, Beijing, China*
[2]*Department of Computer Science, The University of Chicago, Chicago, IL USA*
[3]*Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL USA*
[4] *Computation Institute, The University of Chicago, Chicago, IL USA*
liuwt@act.buaa.edu.cn, kettimut@mcs.anl.gov, libo@act.buaa.edu.cn, foster@anl.gov

Abstract

Data-intensive applications are becoming increasingly common in Grid environments. These applications require enormous volume of data for the computation. Most conventional meta-scheduling approaches are aimed at computation intensive application and they do not take data requirement of the applications into account, thus leading to poor performance. Efficient scheduling of data-intensive applications in Grid environments is a challenging problem. In addition to process utilization and average turnaround time, it is important to consider the worst-case turnaround time in evaluating the performance of Grid scheduling strategies. In this paper, we propose an adaptive scheduling scheme that takes into account both the computational requirements and the data requirements of the jobs while making scheduling decisions. In our scheme, data transfer is viewed in par with computation and explicitly considered when scheduling. Jobs are dispatched to the sites that are optimal in terms of both data transfer time and computation time. In addition, our scheme overlaps a job's data transfer time with its own queuing time and other jobs' computation time as much as possible. Trace-based simulations show that the proposed scheme can gain significant performance benefits for data-intensive jobs.

I. INTRODUCTION

In scientific research communities, more and more large-scale scientific experiments generate and process data on the order of terabytes or even petabytes. For example, four high energy physics experiments at CERN produce and process several petabytes of data per year, and this process is expected to last 15 to 20 years [1]. These "data-intensive applications" require large-scale data storage and computation resources, which typically are geographically distributed. The applications usually need to move data to the computation site for processing. Data movement time can range from a couple of seconds to hours or even days [2].

Grid computing [3] enables users to harness distributed heterogeneous resources to solve complex problems. Metaschedulers [4][15] play a key role in Grid computing, dispatching jobs submitted to the Grid to different sites for execution and monitoring their running.

Conventional metascheduling approaches consider computation requirements alone to schedule jobs and not consider data requirements in which case the data movement should be coupled together with the computation and thus the computational resources need to be allocated during data transfer. This approach has several drawbacks. Typically, the computational sites on the Grid have dedicated nodes for doing data transfers, and a file system that is shared by the computational nodes and the data servers. For example, the individual computational sites on national Grid infrastructures such as TeraGrid, Open Science Grid run GridFTP on dedicated nodes for data transfer and have a shared parallel file system that is accessible to the GridFTP servers as well the computational nodes. Thus, the data transfer does not require any resources on the compute nodes. So, when the metascheduler is not data aware and the data movement is done as part of the computation, the computational resources are essentially wasted for the duration of the data transfer. Also, in Grid environments network conditions vary a lot for different sites. Without knowledge of data requirements, the metascheduler cannot make an optimal selection of the computational resource for the jobs, which leads to poor performance, like low resource utilization, long job turn around time.

In some metaschedulers that are data aware such as GridWay [15] and DAGMan [16], the data is transferred by the metascheduler to a computational site before the job is submitted to the local scheduler. This approach improves resource utilization but has several drawbacks as well. When a computational site is picked, it is picked only based on the computation aspects of the job and not based on the data aspects. A site that will take the longest time (among all the available sites) to transfer the data might be picked. Also, the job is not queued at the local site until the data transfer is done.

We propose an adaptive metascheduling scheme to improve the turnaround time of data-intensive applications and to optimize the resource utilization. There are two key ideas in our scheme: 1) take into account both the computational requirements and the data requirements of the jobs while

making scheduling decisions, and try to get best resources for a job based on that comprehensive consideration; 2) overlap the data transfer time for a job with the queuing time of the job and/or the computation time of other jobs. Specifically speaking, in our scheme, data transfer is viewed equally with computation and explicitly considered when scheduling. Jobs are dispatched to the sites that can finish it earliest based on both the data transfer time and computation time. In addition, as soon as a job is put into the queue of a local scheduler, data transfer is started immediately so that the job's data transfer time is overlapped with its own queuing time and other jobs' computation time to a maximum extent.

The paper is organized as follows. In Section 2, we review previous work related in scheduling of data-intensive applications; In Section 3, the proposed metascheduling scheme is described and analyzed; trace-based simulation results and analysis are presented in Section 4. We present our conclusions and describe future work in Section 5.

II. Related Work

Subramani et al. [5] proposed a greedy meta-scheduling algorithm based on multiple simultaneous requests. The metascheduler identifies the sites that can start the job earliest. This approach is suitable only for homogeneous resources, however, and does not take data requirements into account.

In [6], Wäldrich et al. proposed a meta-scheduler that co-allocates arbitrary types of resources; the goal of this work is to allocate multiple kinds of resource with specific QoS requirements in a certain sequence with consideration of heterogeneity and different site policies. However, this work has no special consideration for data-intensive applications.

Ranganathan et al. [7] developed a family of job-scheduling and data-movement algorithms for data-intensive applications. Data replica is used to improve performance. In this work, computation scheduling and data scheduling (data replication strategies) are independent; they do not incurporate with each other to get best resources for a job.

Chameleon [8] is a resource broker developed for data grid environment. It proposes a family of cost models in terms of job-turnaround time. When a job is submitted, the scheduler uses the cost models to decide how to schedule the job; either the application code or data can be moved to get the best scheduling performance.

Bent et al. [9] discussed the scheduling of a collection of jobs with data requirements. They extended the Condor ClassAd mechanism to enable the worker node to include information on the files available on the node. This work is designed for cluster environments, however; it is not suitable for more distributed Grid computing. In addition, their goal is to maximize throughput while minimizing data movement costs, whereas our goal is to improve job turnaround time.

Venugopal et al. [10] developed a Grid service broker for data-intensive applications. Their objective is similar with that of our work in this paper, and they considered the situation where there are multiple replicas for a data file. The computation resource and data resource pair is also selected in order to minimize job turnaround time. However, they don't consider overlapping the data transfer time with queuing time/computation time.

In job shop scheduling [11][12], both job accounts and machine accounts are finite. Each job is composed of several operations, which must be processed in an orderly fashion. A schedule is a mapping from operations to machines. Job shop scheduling is NP-complete, and heuristics are widely used. Though the data stage-in, computation and data stage-out can be thought of as sequential operations of a data intensive job, multiple data movement operations can be carried out simultaneously on the same resource and thus the scheduling of data intensive jobs does not directly map to job shop scheduling.

In data diffusion [13], when the dispatcher receives a job, it attempts to dispatch this job to a resource that has cached the data needed by the job. The job will first try to get data that is not cached on the execution resource from its peer resources. Only if no cache is available does the job get its data from persistent storage. Often, however, each job requires different data for processing; cached data is not used by subsequent jobs. In that case, jobs have to request data from the data storage site, and the scheduling policy in data diffusion is not able to improve performance much.

Stork [2][14] considered data placement activity a first-class citizen in the Grid. It is capable of queuing, scheduling, monitoring and managing data placement activities. Stork accepts data placement jobs and executes them according to a given scheduling policy (e.g., FCFS, shortest job first, random). Stork is only a data placement scheduler, it interacts with execution planner and batch scheduler for job scheduling. It can not overlap data placement activity with computation activity.

III. Proposed Scheduling Scheme

Our meta-scheduling strategy takes into account not only computation resource requirements but also data requirements (storage space, network condition, etc.) when making scheduling decisions. It ensures that the resource that possesses very powerful processors but bad network conditions to

the data source is not selected. Furthermore, data is staged into the computational site while the job is waiting in the queue, and staged out after the computation resource is released.

Figure 1 illustrates the benefits of the proposed idea from individual job's perspective; it tries to overlap a job's data transfer time with its queuing time. Figure 1(a) shows the conventional execution model in which the scheduler is not data aware. The job waits in the queue until it gets enough resources to execute. Once the job starts, it fetches the input data from a remote location, performs the computation, and transfers the output data to a remote location. Since the data stage-in, stage-out is conducted by the job itself, computational resources are allocated to the job even during the data movement. Thus, the computational resources are wasted during the data transfer. Figure 1(b) shows the approach in which data is staged in before job submission by the metascheduler and staged out by metascheduler after job completion. It can improve resource utilization than the previous one. However, this approach does not take data requirements into account when making scheduling decisions; queuing time is not utilized as well.

Figure 1(c) shows the execution model of our proposed scheme. Here, scheduling decisions are made based on both computation requirements and data requirements. The data stage-in time for a job is overlapped with its own queuing time and with the computation time of other jobs. The data stage-out time is overlapped with the computation time of other jobs. Our scheme attempts to reduce the job turnaround time by decoupling the data stage-in and stage-out from the job execution and at the same time tries to maximize the utilization of computational resources. It calculates the data transfer time for each computational site based on the prevailing available bandwidth and uses that time to select the appropriate resource for the jobs.

The benefits of the proposed scheme from the resource perspective are illustrated in Figure 2. For simplicity, let us assume that all jobs in this illustration require all processors of the computation site. We can see that job2 can conduct its data transfer when job1 is running; job3 can conduct its data transfer when job1 and job2 are running, since now data transfer is not a part of job execution but is done by the scheduler, and the computational resources are not allocated during data transfer. As a result, the turnaround time of job is shortened, and resource utilization is also improved.

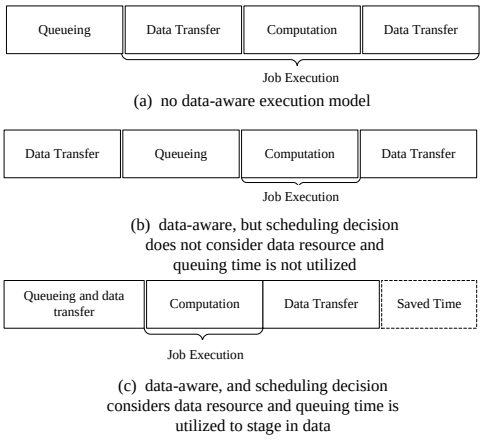


Figure 1. From the individual job's perspective, the proposed scheme can overlap the queuing time and data transfer time.

Figure 3 demonstrates the interaction between the metascheduler and local scheduler. When a job is submitted to the metascheduler, the following information is required: input data location, input data size, output data location, output data size (if known), computational requirements, and estimated runtime. The metascheduler first passes the job description to each local scheduler and queries for an estimate of when the job can be finished. Using the information received from the metascheduler, each local scheduler first estimates available bandwidth from local site to the data source (where the input data required for the computation is located) and to the data sink (where the data generated by the computation needs to be transferred to). This estimate can be obtained either by using historical information or through a network monitoring services such as perfSONAR [23] or NWS [24]. It's notable that due to highly dynamic nature of real network, it's impossible to predict the data transfer time accurately. It's quite possible that data transfer completes earlier or later than expected. We will consider this factor in our future work. Currently, we assume that available bandwidth remains static for the duration of a single transfer. It certainly varies from one transfer to another.

Assume job J is submitted to the metascheduler at time T_0. $D_{stagein}$, $D_{stageout}$ are denoted as input data size and output data size respectively. Let $B_{stagein}$ be the estimated stage-in bandwidth and $B_{stageout}$ be the estimated stage-out bandwidth. Time to stage the data in and out is calculated as

$$T_{stagein} = D_{stagein} / B_{stagein} \quad (1)$$

$$T_{stageout} = D_{stageout} / B_{stageout} \quad (2)$$

Earliest time the job can be started at the local site can be expressed as

$$T_1 = T_0 + T_{stagein} + \delta \quad (3)$$

where δ is the time it takes for the metascheduler to dispatch the job to a local scheduler.
Let T_{start} ($T_{start} >= T_1$) be the guaranteed start time of the job. Then the earliest finish time for the job is

$$T_{finish} = T_{start} + T_{user\_est} + T_{stageout} \quad (4)$$

where T_{user_est} is the estimated run time for the job. T_{finish} is returned to the metascheduler.

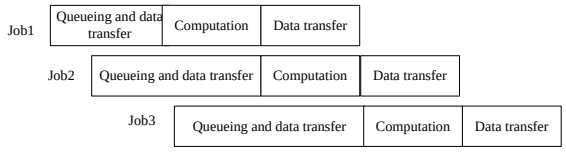


Figure 2. From the resource perspective, the proposed scheme can overlap data transfer time for a job with the computation time of other jobs.

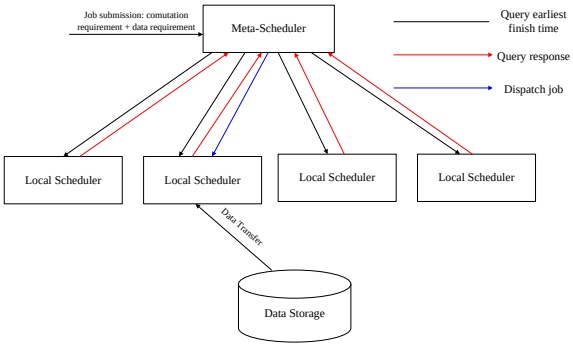


Figure 3. Interaction between the metascheduler and local scheduler.

After collecting a response from each local scheduler, the metascheduler picks the site that can finish the job earliest and dispatches the job to that site. Upon receiving the job, the local scheduler initiates the data movement immediately and makes an advance reservation for the job.

Backfilling [17][18][19] is a widely used and effective optimization approach in scheduling. The backfill algorithm in our scheme takes data movement into account. When a job has a chance to be backfilled, its required data may not have been transferred completely. Under such circumstances, the backfill algorithm should check whether the idle resource hole is big enough to accommodate the computation time and the remaining data transfer time. If it is big enough, the job will be successfully backfilled; otherwise, the job will not be backfilled.

Since our metascheduling scheme values job turnaround time most, it also backfills jobs whose data movement is in progress, in order to have the chance to finish the job as soon as possible. The cost is that some processor time will be wasted. On the other hand, because of the separation of data movement and computation, the processor time required by the job is shorter than that of conventional approaches. Hence, it is easier to find a backfill opportunity, especially for jobs whose computation time is short and data movement time is long. To verify the effectiveness of this backfill policy, we experimentally compared it with a backfill policy that backfills only those jobs that either have no data requirements or the entire data required has been transferred already. The results showed that our backfill policy can get up to 7% improvement in average turnaround time, and only 1% loss in CPU utilization.

IV. Performance Evaluation

In this section we compare our scheme (finish_time_data_aware) with two other meta-scheduling schemes. One scheme is data aware, as is our scheme, and selects the site that can finish the job earliest. It considers the data transfer time when determining the finish time for a job. However, the data transfer and computation still are tightly coupled – there is no overlap between data transfer time and queuing time or computation time. It is referred as "finish_time_no_overlap" in the figures. The other scheme is not data aware and picks the site that can start the job earliest. It is referred to as "start_time_no_data_aware" in the graphs.

A. Experiment Setup

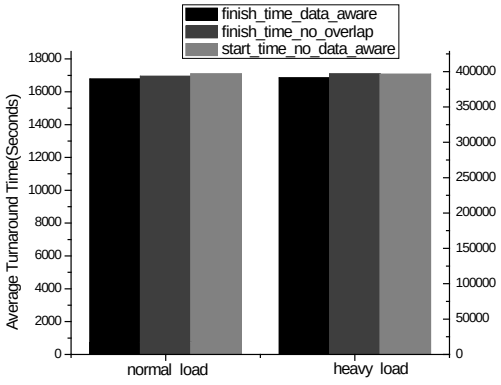
Our performance evaluation uses a locally developed simulator that simulates the behavior of the metascheduler and local scheduler. This simulator incorporated conservative backfill and advance reservation techniques, and also simulates data movement. For simplicity sake, we do not split the data requirements of the jobs into input (stage-in) and output (stage-out) components in the experiments. Rather, we consider all the data requirements as input only.

The three metascheduling schemes are evaluated by using two realistic workload traces from Feitelson's logs [20]. One trace is from the San Diego Supercomputing Center (SDSC) and the other from the Cornell Theory Center (CTC). We use a 5000-job contiguous subset of the traces. The number of processors required for the jobs in CTC and SDSC traces ranged from 1 to 129 and 1 to 128, respectively. The execution times for the SDSC trace ranged from 1 second to 20 hours; for the CTC trace it ranged from 1 second to 4 days.

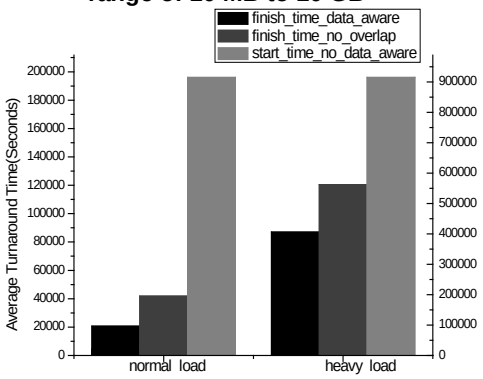
Five computation sites are simulated, having 130, 64, 64, 128, and 32 nodes, respectively. The available bandwidth is calculated as a random value in the range between 10 Mbit/s and 100 Mbit/s. Data transfer time for a job is calculated as

$T_{transfer}$ = data_size / available_bandwidth. Since the traces did not have data requirements for the jobs, we assume the data requirements for the jobs to be between 10 MB and 10 GB for one set of experiments. Since large-scale science applications deal with enormous volumes of data (on the order of terabytes and even petabytes), we evaluated various schemes for a larger data size range from 10 MB to 800 GB. In all experiments, we assume 50% of the jobs have data movement requirements. The proposed scheme is represented as "finish_time_data_aware" in the figures. Simulation studies were performed under both normal and high loads. A high load condition was simulated by increasing the runtime of the jobs by a factor of 3. We use average turnaround time and utilization metrics to compare the three schemes.

B. Evaluation Results



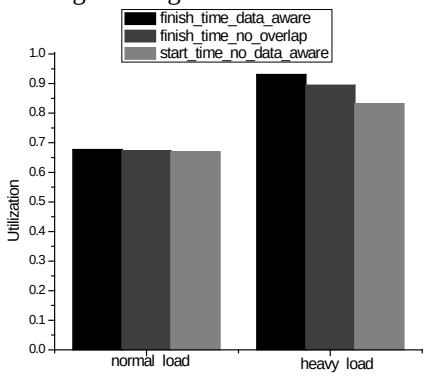
(a) Data requirements of the jobs are in the range of 10 MB to 10 GB



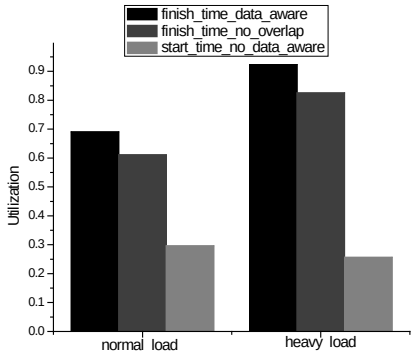
(b) Data requirements of the jobs are in the range of 10 MB to 800 GB

Figure 4. Comparison of average turnaround time for the three schemes – CTC trace.

Figure 4 shows the average turnaround time of different schemes under both normal and heavy load for two different ranges of data requirements. In Figures 4 and 6, the values for the normal load bars is represented in the left y axis, and the values for the high load bars is represented in the right y axis. It is to be noted that only 50% of the jobs have data transfer requirements. We note that our proposed scheme outperforms the other two schemes, but when the data needs of the jobs is in the range of 10 MB to 10 GB, the improvement is not remarkable. The reason is that the computation time is dominant in this situation, and the data transfer time is small relative to the computation time. On the other hand, when the data needs of the jobs are in range of 10 MB to 800 GB, our scheme markedly outperforms the other two schemes. The average turnaround time of our scheme ("finish_time_data_aware") is only about 10% of that of "start_time_no_data_aware" and 50% of that of "finish_time_no_overlap" scheme. The reason is that the data movement time is considerably more and there is much more overlap between queuing time, computation time and data transfer time within our scheme and thus performance gain is significant.



(a) Data requirements of the jobs are in the range of 10 MB to 10 GB



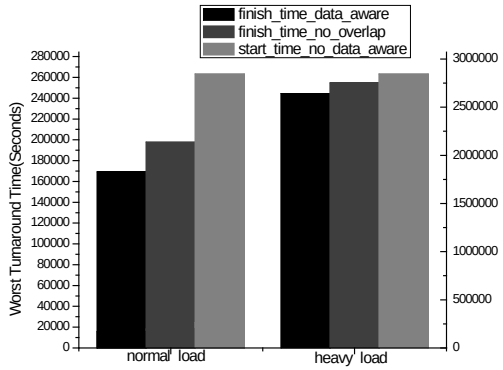
(b) Data requirements of the jobs are in the range of 10 MB to 800 GB

Figure 5. Comparison of processor utilization for the three schemes – CTC trace.

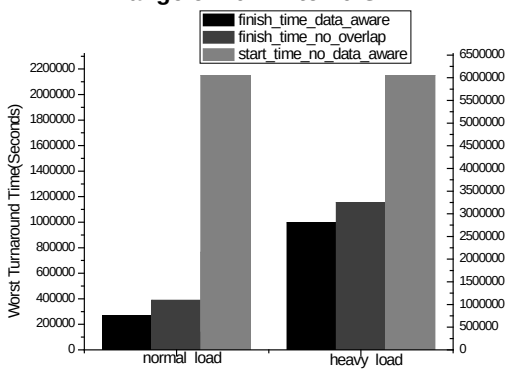
Figure 5 depicts the CPU utilization for the three schemes. Utilization is defined as follows:

(total processing time for all jobs)/ ∑(makespan*number of processors in local computational site). We note that the utilization is better for our scheme, and the difference is significant when the load is heavy or when the data needs are higher.

From Figure 4, we observe that "finish_time-_data_aware" scheme significantly improves the average turnaround time. But from a practical point of view, the worst-case turnaround time is also important. A scheme that improves the average turnaround time but makes the worst-case turnaround time worse is not desirable.



(a) Data requirements of the jobs are in the range of 10 MB to 10 GB



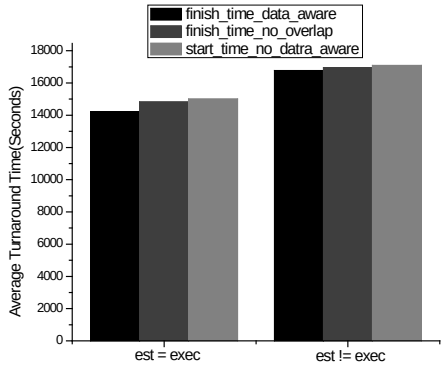
(b) Data requirements of the jobs are in the range of 10 MB to 800 GB

Figure 6. Comparison of worst turnaround time for the three schemes – CTC trace.

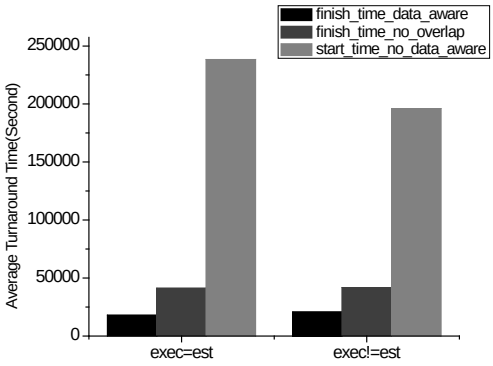
Figure 6 compares the worst-case turnaround time of the three schemes. We note that worst-case turnaround time for our scheme is much better than that for the other two schemes. Since "start_time_no_data_aware" simply picks up a site that is able to start the job earliest, without considering the finish time or the data needs, it may select a site that performs poorly, and thus the worst turnaround time for this scheme is far worse than that for the other two schemes.

C. Impact of Accuracy of User Estimation Runtime

When a job is submitted, the job execution time specified by the user is generally much larger than the time actually needed [21][22]. To evaluate the impact of accuracy of user estimation to job execution time in the context of data-intensive application, we carried out the following experiments. We set the user estimation runtime equals to the actual execution time and compared the results with the previous ones, in which the user estimation is not equal to actual execution time.



(a) Data requirements of the jobs are in the range of 10 MB to 10 GB

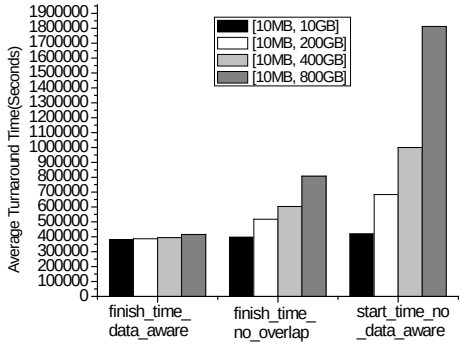


(b) Data requirements of the jobs are in the range of 10 MB to 800 GB

Figure 7. Impact of user-estimation inaccuracy for the three schemes – CTC trace.

Figure 7 compares the performance of the three schemes when the user estimates are inaccurate with the performance when the user estimates are accurate. We see that the performance difference among the three schemes in the inaccurate estimate case is similar to that of the accurate estimate case. Figure 7b also shows that the performance of "start_time_no_data_aware" scheme is better when the user estimates are inaccurate. The reason is that the higher runtime for the jobs (as the data

transfer time is embedded in the job runtime for the "start_time_no_data_aware") causes an increased load, and the inaccurate user-estimate improves the backfilling chances for the smaller jobs especially under heavy load.



.Figure 8. Comparison of average turnaround time for different data sizes with heavy load – CTC trace

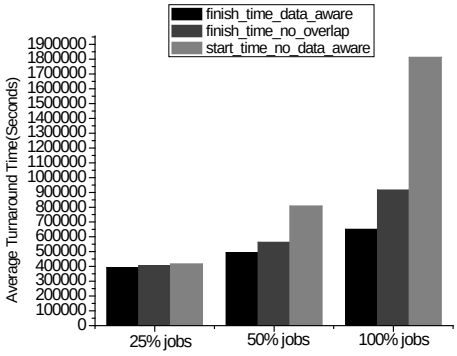


Figure 9. Comparison of average turnaround time of the three schemes for varying proportions of jobs with data requirements – CTC trace.

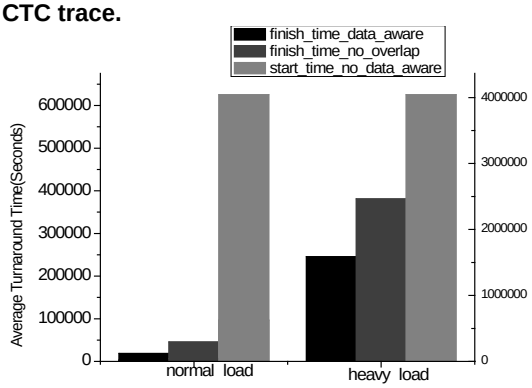


Figure 10. Comparison of average turnaround time for the three schemes – SDSC trace with 50% of the jobs having data requirements in the range of 10 MB to 800 GB.

D. Impact of Varying Data Requirements

Figure 8 compares the performance of the three schemes for varying data transfer requirements. We observe that the average turnaround time for our scheme remains flat for various data requirement ranges. This implies that almost all of the data movement can be overlapped with the queuing time and the computation time for the CTC trace, with 50% of the jobs having data movement needs and the data sizes ranging up to 800 GB.

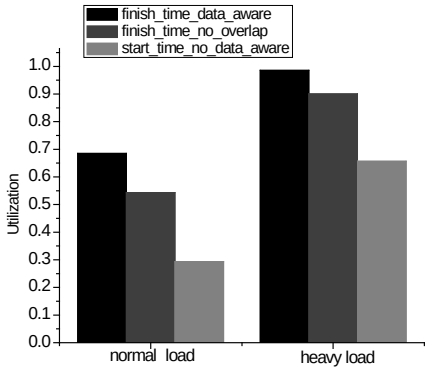


Figure 11. Comparison of CPU utilization for the three schemes – SDSC trace with 50% of the jobs having data requirements in the range of 10 MB to 800 GB.

Figure 9 compares the performance of the three schemes for varying percentage of jobs that require data movement. We note that the average turnaround time with our scheme increases as the percentage of jobs requiring data transfer increases. The performance difference is biggest when all the jobs have data movement needs.

Because of space limitations, we show only the average turnaround time and processor utilization data for the SDSC trace. As noted from Figures 10 and 11, the trends are similar to that of the CTC trace.

V. Conclusions and Future Work

We have explored the issue of scheduling data-intensive jobs using job traces from supercomputer centers. We have proposed an adaptive data-aware scheduling scheme that takes into account not only computation requirements but also data requirements when making scheduling decisions. Moreover, our scheme is able to overlap the data transfer time of a job with its own queuing time and other jobs' computation time. Trace-based simulation demonstrates that our proposed scheme provides significant improvements in average turnaround time and worst-case turnaround time for the jobs. It also provides better utilization of computational resources.

We plan to explore how data transfer failures, common in distributed environments, affect the scheduling decisions of our scheme. We also plan to research workflow applications that consist of multiple data-intensive jobs. The scheduler should schedule the jobs that share data to the same computational site. We intend to incorporate our scheme into a real metascheduler, conduct experiments using real applications, and analyze the results. While doing so, we will integrate our metascheduling scheme with data cache and data replication services in the Grid, in order to optimize the scheduling decision process further.

Acknowledgements. This work was supported in part by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Dept. of Energy, under Contract DE-AC02-06CH11357, and in part by the National Science Foundation's Community Driven Improvement of Globus Software (CDIGS) project.

REFERENCES

[1] W. Hoschek, J. Jaen-Martinez, A. Samar, H. Stockinger, K. Stockinger, Data Management in an International Grid Project, in 2000 Intl. Workshop on Grid Computing (GRID 2000), Bangalore, India, December 2000.

[2] T. Kosar, A New Paradigm in Data Intensive Computing: Stork and the Data-Aware Schedulers, in Proc. of Challenges of Large Applications in Distributed Environments (CLADE 2006) Workshop, in conjunction with HPDC 2006, pp. 5-12, Paris, France, June 2006.

[3] I. Foster, C. Kesselman, and S. Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of High Performance Computing Applications, 15 (3): 200-222, 2001.

[4] Oliver Wäldrich, Philipp Wieder, and Wolfgang Ziegler, A Metascheduling Service for Co-allocating Arbitrary Types of Resources, in Proc. of the 6th International Conference, Parallel Processing and Applied Mathematics, PPAM 2005,, volume 3911 of LNCS, Springer, pages 782-791, Poznan, Poland, September 2005.

[5] V. Subramani, R. Kettimuthu, S. Srinivasan, and P. Sadayappan, Distributed Job Scheduling on Computational Grids Using Multiple Simultaneous Requests, in Proc. of the 11th IEEE Symposium on High Performance Distributed Computing (HPDC 2002), pages 359- 366, Edinburgh, Scotland, July 2002.

[6] O. Wäldrich, P. Wieder, and W. Ziegler, A Metascheduling Service for Co-allocating Arbitrary Types of Resources, pages 782–791 in Parallel Processing and Applied Mathematics, LNCS 3911, Springer Verlag, 2005.

[7] K. Ranganathan and I. Foster, Decoupling Computation and Data Scheduling in Distributed Data Intensive Applications, presented at International Symposium for High Performance Distributed Computing (HPDC-11), Edinburgh, 2002.

[8] S. Park and, J. Kim, Chameleon: A Resource Scheduler in a Data Grid Environment, in Proc. of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid'03), pages 253-260, Tokyo, Japan, 2003.

[9] J. Bent, D. Rotem, A. Romosan, and A. Shoshani, Coordination of Data Movement with Computation Scheduling on a Cluster. In Workshop on Challenges of Large Applications in Distributed Environments (CLADE2005), pages 25–34, Research Triangle Park, NC, July 2005.

[10] S. Venugopal, R. Buyya, and L. Winton. A Grid Service Broker for Scheduling Distributed Data-Oriented Applications on Global Grids, in Proceedings of the 2nd Workshop on Middleware in Grid Computing (MGC 04), Toronto, Canada, ACM Press, Oct. 2004.

[11] A. Jones and L. C. Rabelo, Survey of Job Shop Scheduling Techniques, Technical report, NISTIR, National Institute of Standards and Technology, Gaithersburg, MD, 1998.

[12] Peter Brucker. The Job-Shop Problem: Old and New Challenges, in Proc. of the 3rd Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA), pp. 15-22, Paris, France (28-31 August 2007).

[13] Ioan Raicu , Yong Zhao , Ian T. Foster , and Alex Szalay, Accelerating Large-Scale Data Exploration through Data Diffusion, in Proc. of the 2008 International Workshop on Data-Aware Distributed Computing, pp.9-18, Boston, MA, June 24-24, 2008.

[14] http://www.storkproject.org/

[15] http://www.gridway.org/

[16] http://www.cs.wisc.edu/condor/dagman/

[17] D. Lifka. The ANL/IBM SP Scheduling System, in JSSPP, pages 295–303, 1995.

[18] A. W. Mu'alem and D. G. Feitelson, Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling, in IEEE Transactions on Parallel and Distributed Computing, 12:529–543, 2001.

[19] D. Perkovic and P. J. Keleher. Randomization, Speculation, and Adaptation in Batch Schedulers. Cluster Computing, 3(4):245–254, 2000.

[20] http://www.cs.huji.ac.il/labs/parallel/workload/

[21] Cynthia Bailey Lee, Yael Schwartzman, Jennifer Hardy, and Allan Snavely. Are User Runtime Estimates Inherently Inaccurate? In Proc. of 10th Job Scheduling Strategies for Parallel Processing, June 2004.

[22] Walfredo Cirne and Fran Berman, A Comprehensive Model of the Supercomputer Workload, in Proc. of IEEE 4th Annual Workshop on Job Scheduling Strategies for Parallel Processing. Cambridge, MA. 2001.

[23] perfSONAR: www.perfsonar.net/

[24] NWS: http://nws.cs.ucsb.edu/ewiki/